

1265058

# THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE  
United States Patent and Trademark Office

*December 21, 2004*

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THE UNITED STATES PATENT AND TRADEMARK OFFICE OF THOSE PAPERS OF THE BELOW IDENTIFIED PATENT APPLICATION THAT MET THE REQUIREMENTS TO BE GRANTED A FILING DATE.

APPLICATION NUMBER: 60/523,648  
FILING DATE: *November 20, 2003*  
RELATED PCT APPLICATION NUMBER: *PCT/US04/39380*

Certified by



Jon W Dudas

Acting Under Secretary of Commerce  
for Intellectual Property  
and Acting Director of the U.S.  
Patent and Trademark Office

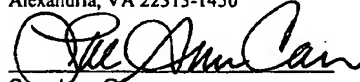
BEST AVAILABLE COPY

U.S. DEPARTMENT OF COMMERCE  
PATENT AND TRADEMARK OFFICE

**PROVISIONAL APPLICATION FOR PATENT  
COVER SHEET**

Address to: Mail Stop Provisional Application  
Commissioner for Patents  
PO Box 1450  
Alexandria, VA 22313-1450  
(703) 308-4357

Certificate Under 37 CFR 1.10  
Date of Deposit: November 20, 2003  
I hereby certify that this correspondence is being  
deposited with the United States Postal Service as  
"Express Mail" service under 37 CFR 1.10 on the date  
indicated above addressed to Mail Stop Provisional  
Application, Commissioner for Patents, PO Box 1450,  
Alexandria, VA 22313-1450

  
Sue Ann Carr  
Express Mail No. ER 319 457 524 US

U.S. PTO  
60/523648  
11/20/03

This is a request for filing a Provisional Application for  
Patent under 37 CFR 1.53(c)

Inventor(s) and Residence(s) (city and either state or foreign country):

Last Name	First Name	Middle Initial	City	State or Country
Buckner	Zach		Charlottesville	Virginia

Title: **METHOD, SYSTEM, AND COMPUTER PROGRAM PRODUCT FOR ENHANCED  
RESOLUTION, AUTOMATICALLY-CALIBRATED MAGNETIC POSITION SENSOR**

20 Sheets of specification.  
3 Sheets of drawings.

University of Virginia Patent Foundation claims small entity status as a nonprofit  
organization (37 CFR §§1.27(a)(3) and (c)). The Commissioner is hereby authorized  
to charge the Small Entity Fee of **\$80** to Deposit Account No. 50-0423.

Please direct all communication relating to this application to:

Robert J. Decker, Esq.  
Patent Counsel  
University of Virginia Patent Foundation  
1224 West Main Street, Suite 1-110  
Charlottesville, VA 22903 U.S.A.

Customer No. 34444  
Telephone: (434) 924-2640  
Fax: (434) 924-2493

This invention was made by an agency of the United States Government or under a contract with  
an agency of the United States Government. The government has certain rights in the invention.

YES ☐ NO ☒ Grant No. \_\_\_\_\_

Dated: November 20, 2003

Respectfully submitted,

By:   
Robert J. Decker (Reg. No. 44,056)

## **Method, System, and Computer Program Product for Enhanced Resolution, Automatically-Calibrated Magnetic Position Sensor**

### **Background**

The present invention method, system, and computer program product provides, among other things, improved magnetic position sensor. Position sensors that use Hall effect elements to detect periodic magnetic discontinuities are well known and have proven useful in industrial applications related to aluminum die casting. Also, several techniques to improve the resolution of these sensors are known. However, existing forms of this sensor suffer from a combination of the following deficiencies, but not limited thereto:

- Many require large chains of analog electronic circuitry, increasing expense and increasing the sensor's sensitivity to electromagnetic noise and environmental changes.
- Many circuits neglect to adequately compensate for changes in amplitude from the Hall effect elements.
- Many systems require manual calibration of analog components like potentiometers.
- Many designs require the size and complexity of the circuitry to scale with the expected resolution enhancement. For example, a further doubling of resolution requires a doubling in size and complexity of circuit components.
- Many designs require operations that are difficult and costly to implement, such as digital division operations.

### **Brief Summary of the invention**

The present invention is a complete magnetic position sensor and related method and computer program code, which overcomes the problems suffered by traditional and improved magnetic positions sensors of this variety. It is capable of increasing resolution by an arbitrary degree, limited only by input signal noise. The sensor has proven capable of resolutions of about .0015 inch, 32 times the resolution of the underlying magnetic stripes, operating at 500

inches per second. The system is small, simple, efficient, and almost entirely digital. It requires minimal analog circuitry and calibrates itself automatically.

## Detailed Description of the Invention

The exemplary individual parts of the invention are described below.

### Hydraulic Ram

A hydraulic ram, with regularly spaced bands of magnetic material around the girth of the ram, is identified in Figure 1. These bands are spaced at intervals of  $1/20^{\text{th}}$  of an inch, although other intervals are possible. In the text that follows, this distance is indicated by the variable *toothPeriod*.

### Hall Effect Sensors

Four Hall effect magnetic sensors, labeled Hall1, Hall2, Hall3, and Hall4 in the Figure 1, are mounted in close proximity to the ram. These sensors are spaced at precise distances with respect to one another, such that the first sensor is at 0 degrees, the second at 90 degrees, the third sensor at 180 degrees, and the fourth sensor at 270 degrees. These angular measurements are relative to the *toothPeriod*, where 360 degrees is equal to one *toothPeriod*, as in the following equation:

$$\text{angle} = 360^{\circ} \times \frac{\text{position}}{\text{toothPeriod}}$$

As the ram moves, the four hall sensors are fixed in position, such that the alternating bands of magnetic material pass beside the Hall effect sensors. Thus the Hall effect sensors provide four time-varying, sinusoidal Hall voltages, with precise phase relationships. If variable *position* is assigned to be the linear position of the ram (where the origin, *position*=0, is assigned to be directly overtop a tooth), then the following relationships exist between p and the hall voltages:

$$h1 = \cos\left(360^{\circ} \times \frac{\text{position}}{\text{toothPeriod}}\right)$$

$$h2 = \cos\left(360^{\circ} \times \frac{\text{position}}{\text{toothPeriod}} - 90^{\circ}\right)$$

$$h3 = \cos\left(360^\circ \times \frac{\text{position}}{\text{toothPeriod}} - 180^\circ\right)$$

$$h4 = \cos\left(360^\circ \times \frac{\text{position}}{\text{toothPeriod}} - 270^\circ\right)$$

### Differential Amplification

Hall voltages h1 and h3 are passed through a differential amplifier (labeled Amp0 in Figure 1), which provides both amplification and common-mode noise rejection. The output voltage for Amp0, labeled a0, follows the equation:

$$a0 = \text{gain} \times (\text{input}_+ - \text{input}_-)$$

Thus, the output can be expressed:

$$a0 = \text{gain} \times \left( \cos\left(360^\circ \times \frac{\text{position}}{\text{toothPeriod}}\right) - \cos\left(360^\circ \times \frac{\text{position}}{\text{toothPeriod}} - 180^\circ\right) \right)$$

Since  $\cos(x + 180^\circ) = -\cos(x)$ , for all x:

$$a0 = 2 \times \text{gain} \times \cos\left(360^\circ \times \frac{\text{position}}{\text{toothPeriod}}\right)$$

Likewise, signals h2 and h4 are connected to Amp90, yielding:

$$a90 = 2 \times \text{gain} \times \sin\left(360^\circ \times \frac{\text{position}}{\text{toothPeriod}}\right)$$

### Digital Controller

The digital controller manages the flow-of-control for the components in the enclosing box in Figure 1. For simplicity, wires that connect the digital controller to the individual components have not been drawn. The invention uses a MSP430 Mixed Signal Processor manufactured by Texas Instruments, although many other choices are available, including Field Programmable Gate Arrays.

### Offset Calibration

The amplified signals, a0 and a90, have an unpredictable DC offset component due to manufacturing tolerances, placement tolerances, and variation in magnetic flux intensity. However, offsets must be corrected to a fixed level of 1.5V to assure that a0 and a90 are

correctly sampled by the analog-to-digital circuitry. precise dc offset is attained using a simple control loop. For signal a0, the control loop consists of the analog-to-digital component ADC0, the averaging component Avg0, the digital-to-analog component DAC0, and the subtractor Sub0. Calibration is performed only when power is first applied to the sensor, a process managed by the digital controller. During calibration, a 0 volt signal is initially outputted by Avg0. Thus, the output from Sub0 follows its input, a0. The ADC0 then samples a0 at 100 kilosamples per second, and Avg0 performs a time-average of these samples over a .25 second interval. The component's output, labeled err0 in Figure 1, is generated by subtracting the resultant average value from one 1.5 volts:

$$\text{err0} = 1.5\text{V} - \text{average}$$

, and outputted digitally by the Avg0 component. At this point, the subtractor will remove this error component from subsequent analog values of a0. Hereafter, the calibrated signal s0 will have a precise DC offset of 1.5 volts. By the same process, signal s90 will also have a correct DC offset of 1.5 volts.

While Avg0 and Avg90 could be implemented as separate external components or internal blocks of an FPGA, this invention implements them in software within the MSP430 digital controller.

### Measurement

The invention contains two analog-to-digital converters, labeled ADC0 and ADC90 in Figure 1, which translate analog signals within the range of 0V to 3V to 12-bit signed integer values within the range -2048 to 2047. After conversion, calibrated signals d0 and d90 vary sinusoidally with respect to the ram position. These signals are centered at a DC offset of 0, the middle of the 12-bit sampling window.

While ADC0 and ADC90 could be implemented as separate external components, this invention uses the analog-to-digital converters that are available on the MSP430 digital controller.

### Signal Selection

After measurement, digital signals d0 and d90 are passed to signal selection component label Sel in Figure 1. This component simply compares the absolute magnitudes of d0 and d90 and returns the signal with the lesser magnitude as its primary output, labeled p in the diagram, and returns the signal with greater magnitude as its secondary output, labeled s in the diagram. The

output signal labeled sel in Figure 1 is a digital signal generated inside the component: it is zero when d0 is the primary signal and one when d90 is the primary signal.

While the signal selection component could be implemented as a separate external component, this invention implements it in software within the MSP430 digital controller.

### Normalization

The normalization component estimates the amplitude of the incoming primary and secondary signals. These signals, p and s, have the form  $\text{amplitude} \times \cos(x)$  and  $\text{amplitude} \times \sin(x)$  or vice-versa. Thus, according to the geometric identity  $\sin^2(x) + \cos^2(x) = 1$ :

$$\sqrt{p^2 + s^2} = \sqrt{\text{amplitude}^2 \times (\cos^2(x) + \sin^2(x))} = \text{amplitude}$$

Rewritten, the equation becomes

$$\text{amplitude} = \sqrt{p^2 + s^2}$$

This appears to require a digital square-root operation. However, since the signals p and s are represented by twelve-bit integers, only 11-bit amplitudes are meaningful. Thus, the operation is greatly simplified through the use of a lookup table. The lookup table works as follows: Elements are stored in a read-only memory with addresses 1,2,... 2048. The value at each memory address is equal to the square of the address. For example the value at address 4 is 16, the value at address 5 is 25. Thus, for a given square x, the task of finding  $\sqrt{x}$  is reduced to a 11-step binary search.

After the amplitude is estimated, the normalization component then scales the primary signal, p according to the following equation:

$$np = \frac{2048 \times p}{\text{amplitude}}$$

, where np is the output of the component, labeled in Figure 1. Again, a lookup table is used to eliminate the costly division operation. The table works as follows: Elements are stored in a read-only memory with addresses 1,2,... 2048, each address corresponding to a possible amplitude as calculated above. The value at each memory address is equal to:

$$\text{value} = \left\lfloor \frac{2048}{\text{address}} \times 2^{10} \right\rfloor$$

The resulting memory value is then multiplied by  $p$  and the result is shifted right by 10 bits. This final ‘right shift’ operation is a fast, digital technique to divide by  $2^{10}$ . Thus, the equation for  $np$  becomes:

$$np = \left\lfloor \frac{2048}{\text{amplitude}} \times 2^{10} \right\rfloor \times p \times 2^{-10} \approx \frac{2048 \times p}{\text{amplitude}}$$

This technique maps signal  $p$ , a value along a sinusoidal signal with *unknown* amplitude, into a value along a sinusoidal signal with *known* amplitude, 2048. While the normalization component could be implemented as a separate external component, this invention implements it in software within the MSP430 digital controller.

### Angle Table

The angle table is a read-only memory that maps sinusoidal input signal  $np$  into a phase angle, a discrete interval within the phase number line. The angle table follows this equation

$$\text{angle} = \left\lfloor \frac{\text{resolution}}{360^\circ} \times \arccos\left(\frac{np}{2048}\right) \right\rfloor$$

, where *resolution* is the number of subintervals within the range  $[0, 360^\circ]$ . Constant *resolution* ultimately dictates the resolution of the sensor; it represents the number of discernable output events for every *toothPeriod*. This invention uses a resolution of 32, although any other resolution constant would suffice.

The equation listed above requires the arccosine operation, a function whose domain is  $[-1, 1]$  and range is  $[0, 180^\circ]$ . The expression  $(np/2048)$  maps the signal  $np$ , which varies from  $[-2048, 2047]$ , to the necessary  $[1, 1]$  interval. The multiplication by  $(\text{resolution} / 360^\circ)$  constant and the subsequent floor operation maps the result of the arccosine operation onto the integer interval  $[0, (\text{resolution}/2) - 1]$ .

Although the above equation describes the nature of the operations, the implementation is greatly simplified through the use of a lookup table, which eliminates the costly arccosine and division operations. The table works as follows: Elements are stored in a read-only memory with addresses  $-2048, -2047, \dots, 2046, 2047$  corresponding to possible values of the signal  $np$ .



The value at each memory address is equal to the result of the entire equation defined above. In other words, the entire equation is reduced to single memory retrieval operation.

While the angle table could be implemented as a separate external component, this invention implements it in software within the MSP430 digital controller.

### Phase Translation

The phase translation component performs two important corrections on signal ang:

- Because the cosine function is not one-to-one, its inverse function arccosine only resolves to the range  $[0, (\text{resolution}/2) - 1]$ , corresponding to the phase interval  $[0, 180^\circ]$ . Thus, a second step is necessary to correctly resolve the phase angle to the interval  $[0, \text{resolution} - 1]$  corresponding to the entire phase interval  $[0, 360^\circ]$ .
- If signal ang is derived from signal d90 (a sinusoid that is 90 degrees ahead of d0), the resulting phase angle must be shifted forward by 90 degrees. Figure 2 summarizes the relationships that exist between d0 and d90.

Both of these corrections are described below, the phase translation component's behavior is broken into several cases:

#### Derived from d0

If signal sel (an output from the signal selection component) is zero, then d0 was selected as the primary signal and ang was derived from d0. As outline earlier, this case requires that signal ang be correctly shifted into the interval  $[0^\circ, 360^\circ]$ . The corrected phase interval can be determined by inspecting the sign of signal d90, which appears at input s. If d90 is greater than 0, then the phase is correctly situated in the  $[0, \pi]$ , as shown in Figure 3. Thus the output, ph1, should match the input, ang:

$$\text{ph1} = \text{ang}$$

If d90 is less than 0, then the phase should be corrected to be in the interval  $[\pi, 2\pi]$ , as shown in Figure 3. The output should be corrected according to the following equation:

$$\text{ph1} = \text{resolution} - 1 - \text{ang}$$

As an example, consider Figure 3. The figure shows normalized sample of 900 appears at signal np. The angle table will map this to a phase angle of  $64^\circ$ , although it is possible that that

the correct angle is  $296^\circ$  (since there are two possible phase angles that correspond to value 900). To determine the correct angle, the value of the secondary signal  $s$  is inspected. This is shown in Figure 4. If the value of  $s$  is positive, then the correct angle is  $64^\circ$ . If it is negative, the correct angle is  $360^\circ - 64^\circ = 296^\circ$ .

### Derived from d90

If  $sel$  equals one, then  $ang$  was derived from  $d90$ . The correct phase interval can be determined by inspecting the sign of signal  $d0$ , which appears at input  $s$ . If  $d0$  is less than zero, then only the 90 degree correction is necessary. This operation is described in the following equation

$$ph1 = \left( ang + \frac{resolution}{4} \right) \% resolution$$

, where the  $\%$  symbol represents modulo division. If  $d0$  is greater than zero, then two corrections are necessary. Signal  $ang$  is first mapped into the interval  $[\pi, 2\pi]$ . Additionally, the signal is shifted forward by 90 degree correction:

$$ph1 = \left( (resolution - 1 - ang) + \frac{resolution}{4} \right) \% resolution$$

After these corrections, signal  $ph1$  correctly represents the phase angle encoded as an unsigned integer between the range  $[0, resolution - 1]$ . While the phase translation component could be implemented as a separate external component, this invention implements it in software within the MSP430 digital controller.

### Phase Register

Following the output phase, described later, that marks the end of each sensing iteration, the output of the phase register ( $ph2$ ) is set to each the output of the phase translation component ( $ph1$ ). Thus, the phase register always holds the phase value from the previous iteration. While the phase register component could be implemented as a separate external component, this invention implements it in software within the MSP430 digital controller.

### Phase Subtractor

This phase subtractor component determines the amount of angular movement for the current iteration, relative to the phase angle from the previous iteration. This movement is simply the difference between the new phase value and the phase value for the previous

iteration, as stored in the phase register. The resulting digital output from this component is the signal labeled *m<sub>re</sub>* in Figure 1. While the phase subtractor component could be implemented as a separate external component, this invention implements it in software within the MSP430 digital controller.

### Overflow Corrector

The overflow corrector handles overflow and underflow cases. These situations arise when the phase angle transitions from 359° to 0° or from 0° to 359°. Such situations yield an apparent movement of -359° or 359°, respectively, though an actual movement of only 1° or -1°, respectively. The following technique corrects this erroneous movement. If the apparent movement, labeled *m* in Figure 1, is greater than (resolution / 2), then *m* is corrected by the equation:

$$mc = m - \text{resolution}$$

If the apparent movement is less than (resolution / 2), then *m* is corrected by the equation:

$$mc = m + \text{resolution}$$

If neither of the above conditions is true, then the input is unmodified:

$$mc = m$$

While the overflow corrector could be implemented as a separate external component, this invention implements it in software within the MSP430 digital controller.

### Output

The output component generates three output signals labeled *lead*, *trail*, and *valid* in Figure 1. These signals are toggled at the end of each sensing iteration. First, the *lead* and *trail* signals are toggled to emulate a traditional quadrature output (which is usually generated by zero crossing detection circuits from sinusoids in quadrature). In this invention, a simple finite state machine controls the lead and trail signals, as described in Figure 5. The signal *m<sub>c</sub>* determines the number of transitions to follow along the state machine, at 100 nanoseconds per state. If *m<sub>c</sub>* is positive, then “forward” transitions are followed; if *m<sub>c</sub>* is negative, then “backward” transitions are followed.

After this signaling is complete, the *valid* signal is raised high for 100 nanoseconds, then returned back to zero. This signal is used to indicate to decoding logic that the output signaling has finished. While the output component could be implemented as a separate

external component, this invention implements it in software within the MSP430 digital controller.

Still other embodiments will become readily apparent to those skilled in this art from reading the above-recited detailed description and drawings of certain exemplary embodiments. It should be understood that numerous variations, modifications, and additional embodiments are possible, and accordingly, all such variations, modifications, sizes, levels, and embodiments are to be regarded as being within the spirit and scope of this document. For example, regardless of the content of any portion (e.g., title, section, abstract, drawing figure, etc.) of this application, unless clearly specified to the contrary, there is no requirement for any particular described or illustrated activity or element, any particular sequence of such activities, or any particular interrelationship of such elements. Moreover, any activity can be repeated, any activity can be performed by multiple entities, and/or any element can be duplicated. Further, any activity or element can be excluded, the sequence of activities can vary, and/or the interrelationship of elements can vary. Accordingly, the descriptions and drawings are to be regarded as illustrative in nature, and not as restrictive.

The invention may be embodied in other specific forms without departing from the spirit or essential characteristics thereof. The foregoing embodiments are therefore to be considered in all respects illustrative rather than limiting of the invention described herein.

## REFERENCES

The following U.S. Patents are hereby incorporated by reference herein in their entirety:

- 5012239,
- 4630928,
- 5442313,
- 5067089,
- 5719789,
- 4587485,

- 4972080
- 3,956,973
- 5,041,784,

In summary, the present invention method, system, and computer program product provides, among other things, a magnetic position sensor for resolution-critical, harsh-environment industrial applications like aluminum die casting. It outperforms existing sensors in every category -- it operates at resolutions up to about 32 times higher than existing sensors at speeds of up to about 500 inches per second. It is inexpensive and can automatically calibrates itself.

The invention is useful for any conceivable position sensing application. For example, it is useful for sensing displacement along pneumatic or hydraulic cylinders. These applications are common in aluminum die casting, medical robotics, forestry and wood manufacturing, etc.

The design of the present invention could be applied to any sensor that provides analog, quadrature signals. Such signals are found in any sensors, most notably optical encoders, etc.

## Appendix: Source Code

The following is C source code, used to program the MSP430F149 mixed signal processor. The source code consists of n files - main.c, cosine.c, and cosineconstants.c.

### main.c

```
#include <msp430x14x.h>
#define COSINERES 32

#define F \
switch(quadrature) { \
    case 0: quadrature = 1; break; \
    case 1: quadrature = 3; break; \
    case 2: quadrature = 0; break; \
    case 3: quadrature = 2; break; \
} \
OUT = quadrature; \
offset++;

#define B \
switch(quadrature) { \
    case 0: quadrature = 2; break; \
    case 1: quadrature = 0; break; \
    case 2: quadrature = 3; break; \
    case 3: quadrature = 1; break; \
} \
P4OUT = quadrature; \
offset--;

void algorithm(void);
long offset;
int zeroA;
int zeroB;
unsigned char quadrature = 0;

void calibrate(void) {
    long i;
    long aveA = 0;
    long aveB = 0;
    long iter = 500000;
```

```

LCDDISPLAY("Calibrating...", 14);
LCDSECONDLIN();
LCDDISPLAY("Move Sensors", 12);

```

```

for (i=0; i<iter; i++) {
    aveA += ADC12MEM0;
    aveB += ADC12MEM1;
}
zeroA = aveA / iter;
zeroB = aveB / iter;

```

```

LDCDCLEAR();

```

```

void main(void) {

```

```

// temporary variable used during calibrations
int calib;

```

```

// only use the watchdog timer if we're using the LCD

```

```

#ifdef LCD
    WDTCTL = WDT_ADLY_1000;           // WDT 1s/4 interval timer
    IE1 |= WDTIE;                     // Enable WDT interrupt
#else
    WDTCTL = WDTPW + WDTHOLD;         // Stop watchdog timer
#endif

```

```

// setup the clock for fast as she'll go

```

```

DCOCTL = DCO0 + DCO1 + DCO2;
BCSCTL1 |= RSEL0 + RSEL1 + RSEL2;

```

```

LCDINITIALIZE();

```

```

// PWM -----

```

```

TACTL = TASSEL1 + TACLR;             // SMCLK, Clear Tar
CCR0 = 4095;                         // PWM Period
CCTL1 = OUTMOD_7;                    // CCR1 reset/set
CCR1 = 2048;                         // CCR1 PWM duty cycle
CCTL2 = OUTMOD_7;                    // CCR2 reset/set

```

```

CCR2 = 2048;                // CCR2 PWM duty cycle
P1DIR |= 0x0C;              // P1.2 and P1.3 output
P1SEL |= 0x0C;              // P1.2 and P1.3 TA1/2 otions
TACTL |= MC0;               // Start Timer_A in up mode

// ADC -----
//

P6SEL |= 0x88;              // Enable A/D channel A0
ADC12CTL0 = ADC12ON+SHT0_2+MSC; // Turn on ADC12, set sampling time
ADC12CTL1 = SHP + ADC12SSEL_2; // Use MCLK for conversion timer.
ADC12CTL1 |= CONSEQ_3;      // Use consecutive mode
ADC12MCTL0 = INCH_7;
ADC12MCTL1 = INCH_3 + EOS;
ADC12CTL0 |= ENC;           // Enable conversions
ADC12CTL0 |= ADC12SC;       // Start conversion

// Port -----
//

P4DIR |= 0xFF;

// Go -----
//

calibrate();

calib = CCR1 - (zeroA - 2048);
if (calib < 0) calib = 0;
if (calib > 4095) calib = 4095;
CCR1 = calib;
calib = CCR2 - (zeroB - 2048);
if (calib < 0) calib = 0;
if (calib > 4095) calib = 4095;
CCR2 = calib;

algorithm();
}

```



**cosine.c**

```

#include "main.h"

#define READ \
    a = ((int) ADC12MEM0) - zeroA;\
    b = ((int) ADC12MEM1) - zeroB; \
    if (a < 0) { aMag = a * -1; } else { aMag = a; } \
    if (b < 0) { bMag = b * -1; } else { bMag = b; }

extern const long div2[];
extern const long squares[];
extern const unsigned char cosine[];
extern unsigned char quadrature;
extern long offset;
extern int zeroA;
extern int zeroB;
long a;
long b;
int aMag, bMag;

void algorithm(void) {
    signed char last = 0;
    signed char delta = 0;
    signed char current = 0;
    int arrayPos;
    long amp = 800;
    long ampSQ;
    long outstanding = 0;

    int l;
    int r;

    while (1) {
        P4OUT ^= 0x10;
        READ;
        ampSQ = a * a;
        ampSQ += b * b;

        // This next block of code performs a cheap square root
        if (ampSQ < squares[1024]) {
            l = 0; r = 1023;

```

```
} else {  
    l = 1024; r = 2047;  
}  
  
if (ampSQ < squares[l+512]) {  
    r -= 512;  
} else {  
    l += 512;  
}  
  
if (ampSQ < squares[l+256]) {  
    r -= 256;  
} else {  
    l += 256;  
}  
  
if (ampSQ < squares[l+128]) {  
    r -= 128;  
} else {  
    l += 128;  
}  
  
if (ampSQ < squares[l+64]) {  
    r -= 64;  
} else {  
    l += 64;  
}  
  
if (ampSQ < squares[l+32]) {  
    r -= 32;  
} else {  
    l += 32;  
}  
  
if (ampSQ < squares[l+16]) {  
    r -= 16;  
} else {  
    l += 16;  
}  
  
if (ampSQ < squares[l+8]) {  
    r -= 8;  
} else {
```

```

    l += 8;
}

if (ampSQ < squares[l+4]) {
    r -= 4;
} else {
    l += 4;
}

if (ampSQ < squares[l+2]) {
    r -= 2;
} else {
    l += 2;
}

if (ampSQ == squares[l]) {
    amp = l;
} else {
    amp = l+1;
}

// We use either signal a or signal b as the "lookup" source,
// depending on which signal is smaller (smaller is better)
if (aMag < bMag) {

    arrayPos = ((a * div2[amp]) >> 10) + 2048;
    if (arrayPos < 0) arrayPos = 0;
    if (arrayPos > 4095) arrayPos = 4095;

    current = cosine[arrayPos];
    if (b < 0) current = COSINERES - 1 - current;

} else {
    arrayPos = ((b * div2[amp]) >> 10) + 2048;
    if (arrayPos < 0) arrayPos = 0;
    if (arrayPos > 4095) arrayPos = 4095;

    current = cosine[arrayPos];
    if (a > 0) current = COSINERES - 1 - current;
    current = (current + (COSINERES / 4)) % COSINERES;
}

delta = current - last;

```

```

// check for overflow or underflow
if (delta > (COSINERES / 2)) {
    delta -= COSINERES;
} else {
    if (delta < -(COSINERES / 2)) {
        delta += COSINERES;
    }
}

switch (delta) {
    case -16: B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; break;
    case -15: B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; break;
    case -14: B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; break;
    case -13: B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; break;
    case -12: B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; break;
    case -11: B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; break;
    case -10: B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; break;
    case -9: B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; break;
    case -8: B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; break;
    case -7: B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; break;
    case -6: B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; break;
    case -5: B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; break;
    case -4: B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; break;
    case -3: B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; break;
    case -2: B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; break;
    case -1: B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; B; break;
    case 1: F; break;
    case 2: F; F; break;
    case 3: F; F; F; break;
    case 4: F; F; F; F; break;
    case 5: F; F; F; F; F; break;
    case 6: F; F; F; F; F; F; break;
    case 7: F; F; F; F; F; F; F; break;
    case 8: F; F; F; F; F; F; F; F; break;
    case 9: F; F; F; F; F; F; F; F; F; break;
    case 10: F; F; F; F; F; F; F; F; F; F; break;
    case 11: F; F; F; F; F; F; F; F; F; F; F; break;
    case 12: F; F; F; F; F; F; F; F; F; F; F; F; break;
    case 13: F; F; F; F; F; F; F; F; F; F; F; F; F; break;
    case 14: F; F; F; F; F; F; F; F; F; F; F; F; F; F; break;
    case 15: F; F; F; F; F; F; F; F; F; F; F; F; F; F; F; break;
    case 16: F; F; F; F; F; F; F; F; F; F; F; F; F; F; F; F; break;

```

00949-01

```
    }

    // raise, then lower the valid line
    P4OUT |= 0x04;
    _NOP();
    P4OUT &= ~(0x04);

    last = current;
}
}
```

## Cosine Constants

Unlike the previous code written in the C language, the following code is written in Java. Once executed, it writes a large collection of tables to the screen or standard output. This resulting text is then saved in a C file called cosineconstants.c. These are the tables necessary to compile cosine.c, listed above.

```
public static void main(String[] args) {
String s = ("unsigned char cosine[] = {");
for (double i=0; i < 4096; i++) {
double y = (i - 2048d) / 2048d; // returns 0 to pi
double x = Math.floor( (Math.acos(y) / Math.PI) * 16d );
s+= ((int) x) + ", ";

// if this is a newline
if (((i + 1) % 64 == 0) & (i != 0))
s += "\n";
}
System.out.println(s);

s = ("float divisors[] = {\n");
for (float i=0; i < 2048; i++) {
String c = "" + (int) ((2048f / i) * 1024);
for (int j = c.length(); j < 10; j++) { c += " "; }
s+= c + ", ";

// if this is a newline
if (((i + 1) % 64 == 0) & (i != 0))
s += "\n";
}
System.out.println(s);

s = ("const long squares[] = {\n");
for (long i=0; i < 2048; i++) {
String c = "" + i*i;
for (int j = c.length(); j < 10; j++) { c += " "; }
s+= c + ", ";

// if this is a newline
if (((i + 1) % 64 == 0) & (i != 0))
s += "\n";
}
System.out.println(s);
```

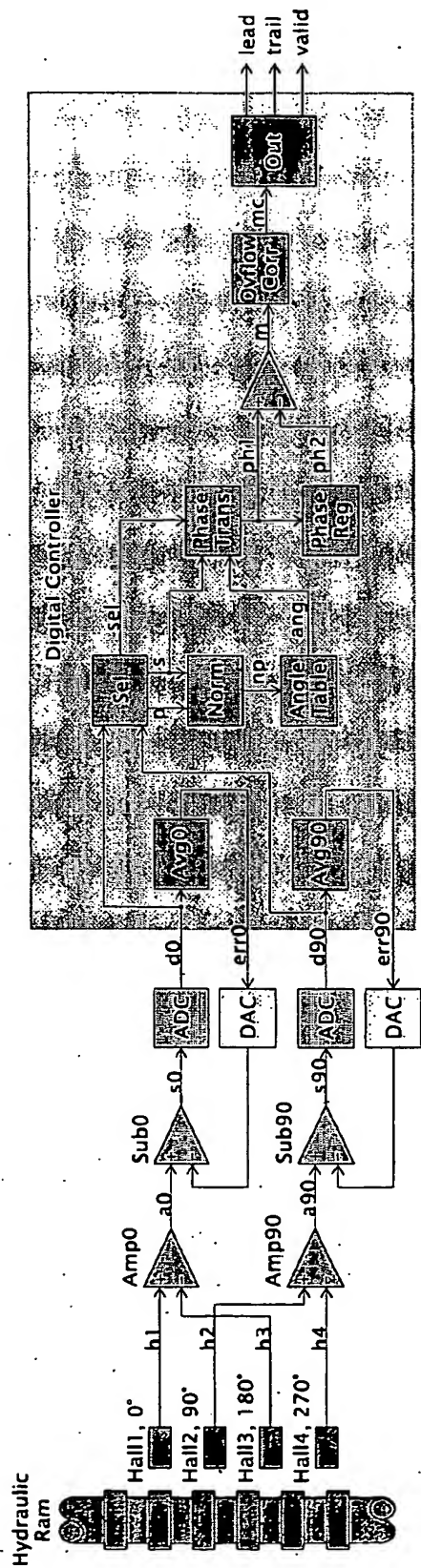


Figure 1: System Parts

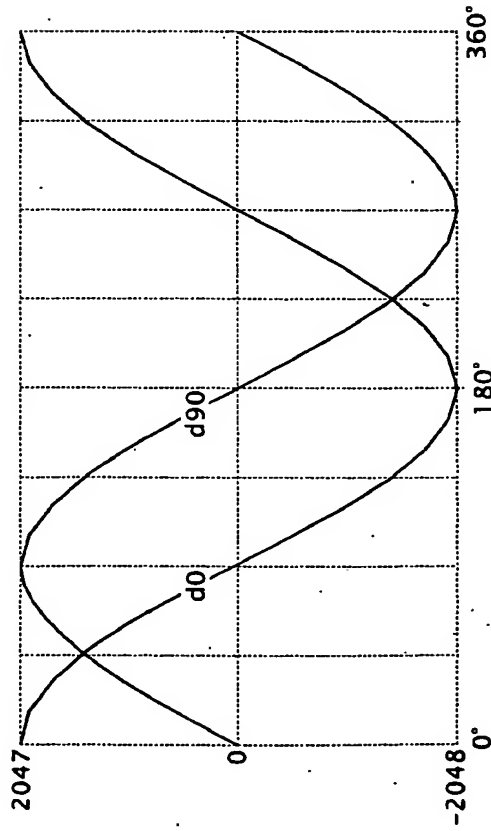


Figure 2: Digital Hall Signals d0 and d90

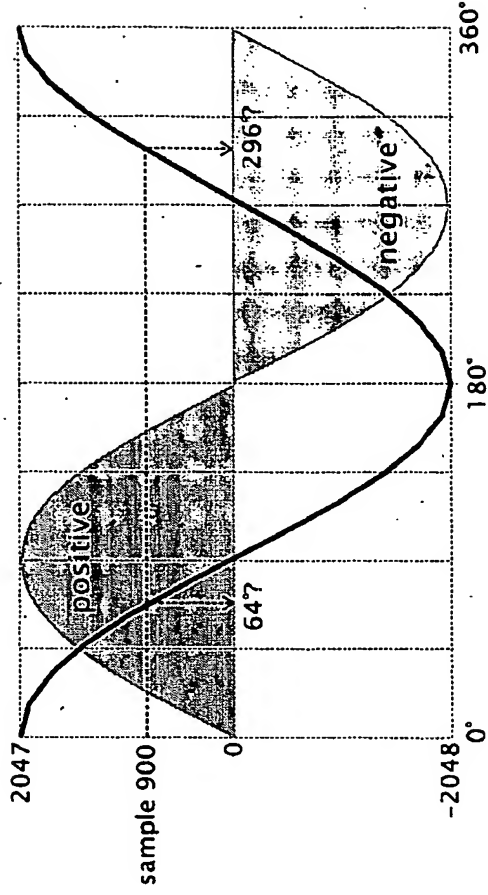


Figure 4: Phase Translation. Resolving the two possible phases for input sample 900 using the sign of the secondary signal.

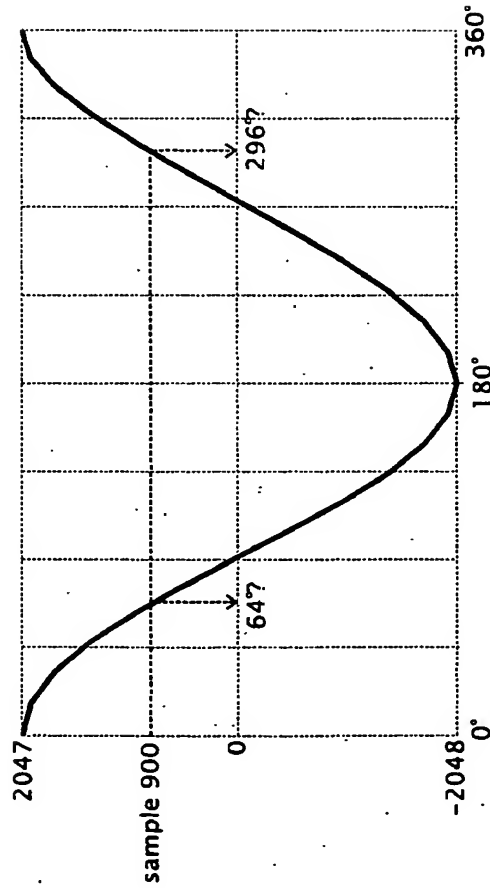


Figure 3: Phase Translation. Two possible phases for input sample 900.

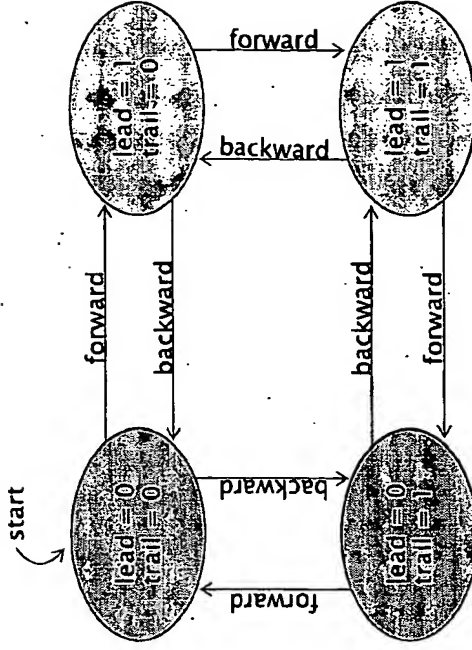
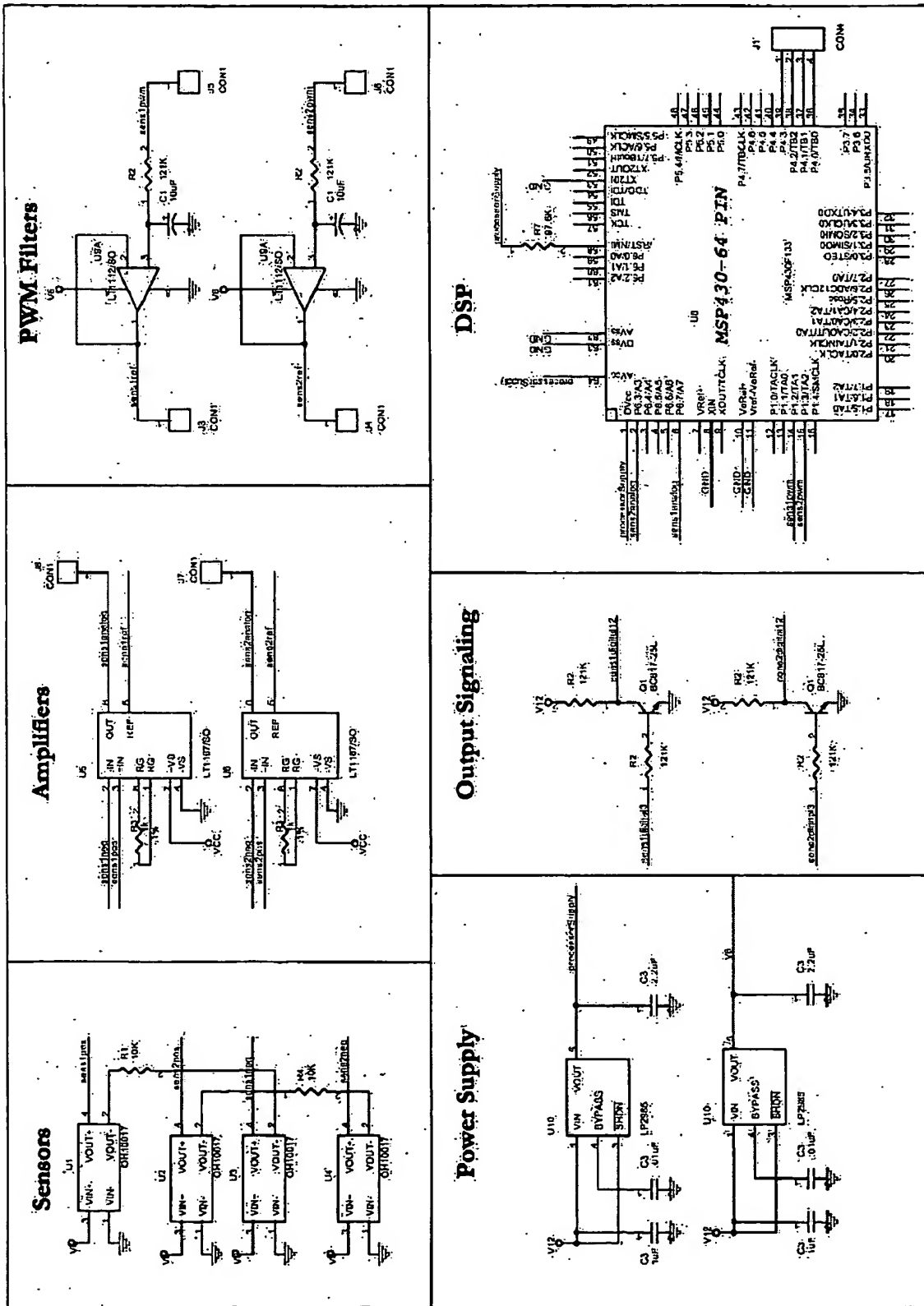


Figure 5: Output Signalling Finite State Machine





# Document made available under the Patent Cooperation Treaty (PCT)

International application number: PCT/US04/039380

International filing date: 22 November 2004 (22.11.2004)

Document type: Certified copy of priority document

Document details: Country/Office: US  
Number: 60/523,648  
Filing date: 20 November 2003 (20.11.2003)

Date of receipt at the International Bureau: 05 January 2005 (05.01.2005)

Remark: Priority document submitted or transmitted to the International Bureau in compliance with Rule 17.1(a) or (b)



World Intellectual Property Organization (WIPO) - Geneva, Switzerland  
Organisation Mondiale de la Propriété Intellectuelle (OMPI) - Genève, Suisse

**This Page is Inserted by IFW Indexing and Scanning  
Operations and is not part of the Official Record**

**BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☒ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: \_\_\_\_\_

**IMAGES ARE BEST AVAILABLE COPY.**

**As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.**